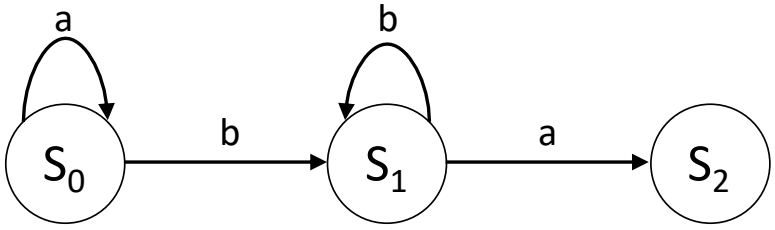
grep: Global Regular Expression Print

Espressioni regolari definiscono dei PATTERN

Consentono di rappresentare sequenze di stringhe che possono essere descritte attraverso un "Automa a stati finiti"



Definisce stringhe costituite da: zero o più 'a' seguite da una (prima freccia) o più 'b' (loop) e completate da una 'a'

- Esempio: "aba", "aaba", "ba", "aaaaaabbbbbbbbba" etc.

`Grep` cerca dei pattern all'interno del file/stream specificato in input

```
================= heroes.txt =================
Catwoman
Batman
The Tick
Spider Man
Black Cat
Batgirl
Danger Girl
Wonder Woman
Luke Cage
The Punisher
Ant Man
Dead Girl
Aquaman
SCUD
Spider Woman
Blackbolt
Martian Manhunter
==============================================
```

- `$ grep -i man heroes.txt`

  Cerca riga per riga, occorrenze di **man** (una **m**, seguita da una **a**, seguita da una **n**) all'interno del file heroes.txt, ignorando (`-i`) la differenza tra maiuscole/minuscole
  Restituisce tutte le righe in cui la ricerca ha successo

  Catwo**man**
  Bat**man**
  Spider **Man**

```
Wonder Woman
Ant Man
Aquaman
Martian Manhunter
```

- `$ grep –v –i man heroes.txt`

  L'opzione `–v` esclude le righe che matchano i criteri di ricerca
  Stampa tutte le righe, eccetto quelle che contengono **man**

```
The Tick
Black Cat
Batgirl
Danger Girl
Luke Cage
The Punisher
Dead Girl
SCUD
Blackbolt
```

Matching per posizione

- `$ grep –E '^Bat' heroes.txt`

  L'opzione `–E` specifica l'utilizzo di una regular expression
  Il carattere ^ indica di fare il match all'inizio della riga
  Usiamo gli apici per forzare la shell a non interpretare ciò che è contenuto tra essi

```
Batman
Batgirl
```

- `$ grep –E 'er$' heroes.txt`

  Il carattere $ indica di fare il match alla fine della riga
  Trova ad esempio:

```
The Punisher
Martian Manhunter
```

  Se avessimo fatto: `$ grep –E 'er' heroes.txt`

```
Spider Man
Danger Girl
Wonder Woman
The Punisher
Spider Woman
Martian Manhunter
```

- `$ grep –E '^$' heroes.txt`

  Combinando ^ e $ di fatto possiamo cercare una linea vuota

- `$ grep –E –i '^(bat|cat)' heroes.txt`

  Con l'operatore | possiamo cercare una istanza, oppure un'altra, all'interno della stessa regexp
  Tutte le righe che iniziano con **bat** o con **cat**, sia maiuscole che minuscole

  Catwoman
  Batman
  Batgirl

- `$ grep –E '^[BbCc]at' heroes.txt`

  Con le parentesi `[ ]` possiamo specificare un insieme di caratteri da matchare

- `$ grep –E '^[A-Z]at' heroes.txt`

  Se all'intero delle parentesi utilizziamo – possiamo specificare intervalli

`[A-Z]`	tutti i caratteri alfabetici maiuscoli
`[a-z]`	tutti i caratteri alfabetici minuscoli
`[A-Za-z]` oppure `[A-z]`	tutti i caratteri alfabetici maiuscoli e minuscoli
`[A-z0-9_]`	come sopra, più i numeri
`[A-MXYZ]`	tutti i caratteri maiuscoli tra A ed M, e anche X, Y e Z

- `$ grep –E –i '[^b]at' heroes.txt`

  Se all'interno delle parentesi utilizziamo ^ possiamo escludere i caratteri che seguono
  `'[^b]at'` : Tutte le righe in cui compare **at**, non preceduto da **B** o **b** : esclude **Bat** o **bat**

  Catwoman
  Black **Cat**

Caratteri ripetuti

Supponiamo di volere individuare username di al più 8 caratteri, che iniziano con una lettera e contengono lettere o numeri.
Una possibile espressione regolare potrebbe essere:

`[a-z][a-z0-9][a-z0-9][a-z0-9][a-z0-9][a-z0-9][a-z0-9][a-z0-9]`

Ma è complessa da scrivere, e non individuerebbe username con un numero di caratteri < 8

Attraverso le parentesi `{}` possiamo specificare il numero di volte in cui vogliamo che un carattere possa ripetersi.

Ad esempio

`^[A-z][A-z0-9]{2,7}$`

Fa il matching di tutte le stringhe che iniziano con una lettera (`^[A-z]`), seguita tra 2 e 7 volte da una lettera o un numero (`[A-z0-9]{2,7}`) prima della fine della riga (`$`)

In generale, `X{n,m}` fa il matching di almeno `n` e non più di `m` ripetizioni del carattere `X`
Omettendo `m`, l'espressione fa il matching di almeno `n` ripetizioni di `X`

Esempi:

`{2}` esattamente due occorrenze del carattere precedente
`^G[o]{2,}gle` descrive `Gooogle`

`{2,}` almeno due occorrenze del carattere precedente
`^G[o]{2,}gle` descrive `Google`, `Gooogle`, `Goooogle` ...

`{0,1}` zero o una occorrenza del carattere precedente, si abbrevia con `?`
`Goo?gle` identifica sia `Gogle` che `Google`

`{1,}` almeno una occorrenza del carattere precedente, si abbrevia con `+`
`Goo+gle` identifica sia `Google` che `Gooogle` che `Goooogle`...

`{0,}` zero o più occorrenze del carattere precedente, si abbrevia con `*`
`Goo*gle` identifica sia `Gogle` che `Google` che `Google`...

Alcuni operatori di uso comune:

`.`	*Match any single character.*
`^`	*Match the empty string that occurs at the beginning of a line or string.*
`$`	*Match the empty string that occurs at the end of a line.*
`A`	*Match an uppercase letter A.*
`a`	*Match a lowercase a.*
`\d`	*Match any single digit.*
`\D`	*Match any single non-digit character.*
`\w`	*Match any single alphanumeric character; a synonym is [:alnum:].*
`[A-E]`	*Match any of uppercase A, B, C, D, or E.*
`[^A-E]`	*Match any character except uppercase A, B, C, D, or E.*
`X?`	*Match no or one occurrence of the capital letter X.*
`X*`	*Match zero or more capital Xs.*
`X+`	*Match one or more capital Xs.*
`X{n}`	*Match exactly n capital Xs.*
`X{n,m}`	*Match at least n and no more than m capital Xs. If you omit m, the expression tries to match at least n Xs.*
`(abc|def)+`	*Match a sequence of at least one abc and def; abc and def would match.*

Esempi:

- `$ cat demo.txt`

```
foo.txt
bar.txt
foo1.txt
bar1.doc
foobar.txt
foo.doc
bar.doc
dataset.txt
purchase.db
purchase1.db
purchase2.db
purchase3.db
purchase.idx
foo2.txt
```

- `$ grep 'purchase' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase

```
purchase.db
purchase1.db
purchase2.db
purchase3.db
purchase.idx
```

- `$ grep 'purchase.' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase seguito da un altro qualsiasi carattere (.)

```
purchase.db
purchase1.db
purchase2.db
purchase3.db
purchase.idx
```

- `$ grep 'purchase.db' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase seguito da un altro qualsiasi carattere (.) e poi da db

```
purchase.db
```

- `$ grep 'purchase..db' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase seguito da due qualsiasi altri caratteri (..) e poi da db

```
purchase1.db
purchase2.db
purchase3.db
```

- `$ grep 'purchase.\.' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase seguito da un qualsiasi altro carattere e poi da un punto (il carattere \ funge da escape)

  ```
  purchase1.db
  purchase2.db
  purchase3.db
  ```

- `$ grep –i –E '^purchase[0–9]?\.' demo.txt`

  Tutti i file contenuti all'interno di demo.txt il cui nome inizia con purchase seguito da zero o più occorrenze di un numero, e poi da un punto

  ```
  purchase.db
  purchase1.db
  purchase2.db
  purchase3.db
  purchase.idx
  ```

Tutti gli esempi precedenti possono essere riscritti utilizzando la pipe

```
$ grep –i –E '^purchase[0–9]?\.' demo.txt
```

Equivale a:

```
$ cat demo.txt | grep –i –E '^purchase[0–9]?\.'
```

Tutte le opzioni

- **`-i`**: ignore the case of your search term
- **`-v`**: show lines that *don't* match, instead of those that do
- **`-c`**: instead of returning matches, return the *number* of matches
- **`-x`**: return only an exact match
- **`-E`**: interpret search as an extended regular expression
- **`-F`**: interpret search as a list of fixed strings, including newlines, dots, etc
- **`-f`**: get the search patterns from this file
- **`-H`**: print the filename with each match
- **`-m`**: stop reading file after *n* number of matches
- **`-n`**: print the line number of where matches were found
- **`-q`**: don't output anything, but exit with status 0 if any match is found (check that status with `echo $?`).
- **`-A`**: print *n* number of lines after the match
- **`-B`**: print *n* number of lines before the match
- **`-o`**: print only the matching part of the line
- **`-e`**: search literally, and protects patterns starting with a hyphen
- **`-w`**: find matches surrounded by space
- **`--color`**: add color to the matched output
- **`--help`**: get some help
- **`-V`**: get grep's version

Esempi:

- `$ cat file.txt`

```
Stewart
Christina
- 441
<a href="https://www.google.com">Google</a>
 Jill
54r4h
Shazbot123
lll
221
Item 1, Item 2, Item 3
TABS    TABS    TABS
23.44.124.67
172.16.23.1
```

- `$ cat file.txt | grep –i jill`

  Cerca jill all'interno di file.txt ignorando maiuscole/minuscole.
  Restituisce:

```
 Jill
```

- `$ cat file.txt | grep –i –E '^jill'`

  Cerca jill all'inizio di una riga di file.txt ignorando maiuscole/minuscole
  Restituisce un risultato nullo, poiché l'unica occorrenza di jill è preceduta da uno spazio

- `$ cat file.txt | grep –i –E '^.?jill'`

  Cerca jill all'inizio di una riga di file.txt ignorando maiuscole/minuscole, ma specifica che prima di jill possono essere presenti 0,1 caratteri.
  Restituisce:

```
 Jill
```

- `$ cat file.txt | grep –v Christina`

  Restituisce tutte le righe che non matchano l'espressione specificata:

```
Stewart
Christina
- 441
<a href="https://www.google.com">Google</a>
 Jill
54r4h
Shazbot123
lll
221
Item 1, Item 2, Item 3
TABS    TABS    TABS
23.44.124.67
172.16.23.1
```

- `$ cat file.txt | grep -E '[[:digit:]]'`

  Restituisce tutte le righe che contengono un numero:

  ```
  - 441
  54r4h
  Shazbot123
  221
  Item 1, Item 2, Item 3
  23.44.124.67
  172.16.23.1
  ```

- `$ cat file.txt | grep -E '^[[:digit:]]'`

  Restituisce tutte le righe che iniziano con un numero:

  ```
  54r4h
  221
  23.44.124.67
  172.16.23.1
  ```

  Equivale a `cat file.txt | grep -E '^[0-9]'`

- `$ cat file.txt | grep -E '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'`

  Restituisce tutte le righe che contengono un indirizzo IP:

  ```
  23.44.124.67
  172.16.23.1
  ```

- `$ cat file.txt | grep -n -E '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'`

  Restituisce tutte le righe che contengono un indirizzo IP e specifica il numero di riga:

  ```
  12:23.44.124.67
  13:172.16.23.1
  ```

AWK - pattern-directed scanning and processing language

`Grep` individua dei pattern all'interno di un file/stream

`AWK` consente di individuare pattern all'interno di file/steram e modificare i contenuti di file/stream

Consente di definire delle **`action`** da effetture quando viene individuato un `pattern`

Come grep opera per righe di input

Ogni linea di input viene processata come se fosse formata da un insieme di campi separati da spazio, o da un field separator `FS` che può essere specificato con il parametro `–F`

- `$ cat file2.txt`

```
CREDITS,EXPDATE,USER,GROUPS
99,01 jun 2018,sylvain,team:::admin
52,01 dec 2018,sonia,team
52,01 dec 2018,sonia,team
25,01 jan 2019,sonia,team
10,01 jan 2019,sylvain,team:::admin
8,12  jun 2018,öle,team:support
```

- `$ awk '1 {print}' file2.txt`

  La coppia `'1 {print}'` è uno statement del tipo `'pattern {action}'`
  Se per una certa riga il `pattern` restituisce TRUE, allora viene eseguita l'`action`
  In questo caso viene eseguita l'action `print`, che è anche l'action di default
  Il comando precedente equivale quindi a:
  `awk 1 file2.txt`

  Se non viene specificato alcun pattern, il match viene fatto su qualsiasi istanza di input

  `awk {print} file2.txt`

Per ciascuna riga, ciascun campo viene identificato con `$1`, `$2`, … $NF
L'intera riga corrisponde al campo `$0`

- `awk '{print $0}' file2.txt`

  Stampa tutte le righe

- `awk '{print $1}' file2.txt`

  Stampa il primo campo di tutte le righe

```
CREDITS,EXPDATE,USER,GROUPS
99,01
```

```
52,01
52,01
25,01
10,01
8,12
```

- `awk -F , '{print $1}' file2.txt`

  Stampa per tutte le righe il primo campo delimitato dal nuovo carattere separatore `,`

```
CREDITS
99
52
52
25
10
8
```

- `awk -F , '{print $1,$3}' file2.txt`

  Stampa per tutte le righe il primo ed il terzo campo delimitato dal carattere separatore `,`

```
CREDITS USER
99 sylvain
52 sonia
52 sonia
25 sonia
10 sylvain
8 öle
```

- `awk -F , '{print $NF}' file2.txt`

  Stampa per tutte le righe l'ultimo campo delimitato dal carattere separatore `,`

```
GROUPS
team:::admin
team
team
team
team:::admin
team:support
```

Esempi con pipe ed ls

- `$ ls -la | awk '{print $1}'`

  Stampa il primo campo restituito da `ls -la` utilizzando il separatore di default (spazio)

Se il pattern è individuato da una espressione regolare, si utilizza la sintassi
`awk '/regex/' filename`

- `$ ls –la | awk '/^d/'`

  Stampa tutti i campi (action di default) delle righe che iniziano con una d (le directory)

- `$ ls –la | awk '/^[–d]/'`

  Stampa tutti i campi (action di default) delle righe che iniziano con – (file) o d (directory)

- `$ ls –la | awk '/^[–]/ {print $9}'`

  Stampa il campo numero 9 (il filename) delle righe che iniziano con – (file)

  Per indicare il pattern deve essere soddisfatto su un campo specifico si usa la tilde

- `$ ls –la | awk '$7 ~ /^A/ {print $9}'`

  Stampa il campo numero 9 (il filename) di tutte le righe il cui campo numero 7 (mese di modifica) inizia per A

  Possiamo applicare più espressioni regolari su diversi campi contemporaneamente

- `ls –la | awk '$7 ~ /^A/ && $1 ~ /^–/ {print $NF}'`

  Come l'esempio precedente, ma usiamo $NF invece di $9 e stampiamo soltanto i file

SCRIPT

La prima linea di uno script è

```
#!/bin/sh
```

Il carattere # indica un commento ed è necessario per dire allo specifico interprete dello script che si vuole eseguire di ignorare la prima linea

La combinazione `#!` si chiama SHEBANG, o HASHBANG, ed indica al sistema quale interprete utilizzare per eseguire lo script stesso

Altri interpreti:

```
#!/usr/bin/perl   <--perl script'
#!/usr/bin/php <-- php script
```

Esempio:

```
#!/bin/sh
clear
echo "HELLO WORLD"
```

Di default lo script viene creato senza permessi di esecuzione

```
$ ls -l temp.sh
-rw-r--r--  1 marcomorana  staff  0  8 Apr 16:44 temp.sh
```

Possiamo modificare i permessi per renderlo eseguibile da tutti (user, group, owner)

```
$ chmod +x temp.sh
$ ls -l temp.sh
-rwxr-xr-x  1 marcomorana  staff  0  8 Apr 16:44 temp.sh
```

Gli script si eseguono semplicemente richiamandoli da terminale

```
$ /path/to/yourscript.sh
```

Se siamo già nella directory in cui si trova lo script possiamo utilizzare il `.` (directory corrente)

```
$ ./yourscript.sh
```

Possiamo passare argomenti di input agli script
All'interno dello script, possiamo accedere ai diversi argomenti con la seguente sintassi

- `$#` Numero di argomenti
- `$*, $@` Tutti gli argomenti passati alla shell
- `$1, $2, ...` Argomento di posto 1, 2, ...
- `$0` Argomento di posto 0, nome del comando stesso
- `$-` Opzioni passate alla shell
- `$?` L'ultimo comando eseguito
- `$$` PID della shell

Esempio:

```
#!/bin/bash
echo "num argomenti di input: $#"
```

- `$ ./s1.sh a b c d`

```
#!/bin/bash
echo "argomenti di input: $@"
```

- `$ ./s2.sh a b c d`

```
#!/bin/bash
echo "primo argomento di input: $1"
```

- `$ ./s3.sh a b c d`

```
#!/bin/bash
for arg in "$@"
do
    echo "$arg"
done
```

- `$ ./s4.sh a b c d`

Usare gli argomenti di input per definire filtri parametrici

- `$ ls –la | awk '$6 ~ /^a/ {print $9}'`

  Stampa il campo numero 9 (il filename) di tutte le righe il cui campo numero 6 (mese di modifica) inizia per a (aprile)

  ```
  #!/bin/bash
  ls –la | awk –v arg="$1" '$6 ~ "^" arg {print $9}'
  ```

  Stampa il campo numero 9 (il filename) di tutte le righe il cui campo numero 6 (mese di modifica) inizia con la lettera specificata come parametro di input
  Il carattere ^ (inizio stringa) viene concatenato ad arg utilizzando " "
  `$ ./s5.sh m`

Verificare il numero di argomenti di input – costrutto IF

String Comparison	Description
Str1 = Str2	Returns true if the strings are equal
Str1 != Str2	Returns true if the strings are not equal
-n Str1	Returns true if the string is not null
-z Str1	Returns true if the string is null
Numeric Comparison	**Description**
expr1 -eq expr2	Returns true if the expressions are equal
expr1 -ne expr2	Returns true if the expressions are not equal
expr1 -gt expr2	Returns true if expr1 is greater than expr2
expr1 -ge expr2	Returns true if expr1 is greater than or equal to expr2
expr1 -lt expr2	Returns true if expr1 is less than expr2
expr1 -le expr2	Returns true if expr1 is less than or equal to expr2
! expr1	Negates the result of the expression
File Conditionals	**Description**
-d file	True if the file is a directory
-e file	True if the file exists (note that this is not particularly portable, thus -f is generally used)
-f file	True if the provided string is a file
-g file	True if the group id is set on a file
-r file	True if the file is readable
-s file	True if the file has a non-zero size
-u	True if the user id is set on a file
-w	True if the file is writable
-x	True if the file is an executable

```
#!/bin/bash

if [ $# -eq 0 ]
then
    echo "Specificare un parametro in input"
elif [ $# -gt 1 ]
then
    echo "Troppi parametri di input"
else
    ls -la | awk -v arg="$1" '$6 ~ "^" arg {print $9}'
fi
```

- `$ ./s6.sh` (eseguire senza parametri, con due parametri, con un parametro)

Altri costrutti

CASE

case *expression* **in**
  pattern1)
    statements ;;
  pattern2)
    statements ;;
  ...
esac

Esempio:

```
#!/bin/bash

case $1 in
     -a | -A | --aplha )
          echo "alpha" ;;
     -b )
          echo "bravo";;
     -c )
          echo "charlie";;
     * )
          echo "opzione sconosciuta";;
esac
```

WHILE

while *conditionlist* **do**
    lista comandi
done

Esempio:

```
#!/bin/bash

counter=$1
factorial=1
while [ $counter -gt 0 ] do
     factorial=$(( $factorial * $counter ))
     counter=$(( $counter - 1 ))
done
echo $factorial
```